Resource Management Using Multiple Feedback Loops

V. Kalogeraki, P.M. Melliar-Smith, L. E. Moser
Department of Electrical and Computer Engineering
University of California, Santa Barbara

http://alpha.ece.ucsb.edu/~vana
vana@alpha.ece.ucsb.edu
Motivation

- Size, complexity and heterogeneity of distributed applications are rapidly increasing
- Distributed applications
  - Subject to variations in resource availability, processing speed and timing constraints
  - Require real-time and QoS guarantees
  - Are replicated for fault tolerance or high availability
- Many allocation algorithms generate preplanned schedules for the activities
- No scheduling algorithm is optimal without a priori knowledge of deadlines, computation times and arrival times of the activities
What Is Needed

Resource Management System
- Maximize number of activities that meet their deadlines
- Provide end-to-end QoS and timing guarantees to the activities
- Dynamically deploy the objects to processors
- Balance the load on the processors
- Schedule driven by urgency of activities and importance of activities and objects
The Resource Management system uses a three-level feedback loop with different levels of granularity.
Information Base

Activity Metrics
- Deadline
- Importance
- Projected Latency
- Residual Laxity
- Method Invocation Graph

Object Metrics
- Importance
- Mean processing time
- Replication Degree and Type
The Challenge

- Can we guarantee that activities will meet their deadlines, given object dependencies and resource requirement constraints?
- How can we monitor the actual behavior of the objects on the processors and the actual usage of the resources?
- How should we respond to transient changes in the load or the availability of the resources?
Resource Manager

- Maintains global view of the system
- Distributes objects across multiple processors
- Tries to balance the load on the resources
- Determines Replication Degree and Type of objects based on
  - Importance of activities in the system
  - Importance of object groups invoked by the activities
  - Calculates the utility of the objects in the system
Run-Time Monitoring

- Profilers supply feedback to Resource Manager
- Profiler Measurements
  - Current utilization of processor resources (cpu, memory, disk)
  - Bandwidth on the communication links
  - Usage of resources
  - Number of method invocations
  - Method processing and communication times
  - Residual laxity for activity on completion of the activity
Method Invocations

![Diagram of client and server processes with method invocations and associated times]

Vana Kalogeraki  
University of California, Santa Barbara
Dynamic Scheduling Algorithm

- Least Laxity Scheduling doctrine

\[
Laxity_{valuen} = Deadline_{net} - Projected_{laten}y_{n}
\]

- Laxity_value is carried with each method invocation

- Global Scheduler determines whether new object can be scheduled
  - Computes the effects of increased load on the latencies of existing tasks
  - Computes the contributions to the processor’s queueing length as a result of the addition of the new object
Local Scheduler

- Maintains ready queue with objects to be scheduled on processor
- Uses object’s laxity value to determine if object can be scheduled on processor
  \[ \sum_{0 \leq j < i-1} CompTime_j \leq L_i \]
- Uses object’s importance to move object to an earlier position in the queue
- Computes real-time priority of object
  \[ rt\_priority_i = \frac{c}{laxity_i} + \frac{1}{imp_i} \]
Local Dispatcher

- Selects object located at the front of ready queue
- Object’s realtime priority sets the priority of thread in which it is dispatched
- Thread priorities are mapped into specific realtime priorities of the operating system
- For example, Solaris RT scheduling class and `priocntl` to manipulate scheduling priorities
Residual Laxity

- Recorded as the remaining laxity of the activity
- Compared with the activity’s initial laxity
- Indicates if the projected latency was good estimate of activity’s computation time
- Use ratio of the residual laxity to the initial laxity of the activity to adjust the estimates of projected latency
Resource Management Control

![Diagram of resource management control process]

- Projected task processing profile: graph of object invocations
- Object processing and communication profile
- Projected object invocation latencies
- Resource performance and load
- Residual laxity
- Measured processing and communication times
- Scheduling and processing

First Level of Feedback Loop (milliseconds)
Second Level of Feedback Loop (seconds)
Arrival of New Application Tasks

- The Resource Manager tries to accommodate all activities in the system
- Addition of a new activity can result in
  - Degradation of performance of existing activities
- Action taken by the Resource Manager
  - Reduction of the degree of object replication
  - Migration of objects to different processors
Cooling Algorithm

- Employed when observed load on processor is high
- Most overloaded (high) and least overloaded (low) processors are selected
- Object Load for each object on high computed as:

\[ ObjectLoad_i = \sum_t \sum_m \tau_{mp} x_{tm} \times MeanInvoc_t \]

- Object with the highest load is migrated from the most to the least overloaded processor
Hot Spot Algorithm

- Invoked when estimated projected latency for an activity is high
- Queueing latency for each activity computed as:

\[
Queuing\ Latency_{tip} = \sum_{m \in i} \frac{x_{tm} \tau_{mp}}{1 - \rho_p} - x_{tm} \tau_{mp}
\]

- Object whose methods cause the largest increase in the latency is migrated to the least overloaded processor (low)
Object Allocation and Reallocation

New application task

Tentative allocation of objects to processors

Projected task processing profile: graph of object invocations

Projected task latency

Latency too high

Second Level of Feedback Loop (seconds)

Third Level of Feedback Loop (several seconds)

Object processing and communication profile

Projected object invocation latencies

Projected load or latency is too high

Loss of processor

Resource performance and load

Load too high

Reallocate objects or reduce degree of replication
Conclusion and Future Work

- Complex distributed object systems require profiling, scheduling and migration algorithms
- Evaluate effectiveness of multiple feedback loops
- Experiment with VxWork operating system
- Experiment with different ORBs
- Investigate several different real-time distributed object-oriented applications